1、ES6：map

ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合。但是“键” 的范围不限于字符串，各种类型的值（包括对象）都可以当作键。Map 也实现了 iterator 接口，所以可以使用『扩展运算符』和『for...of...』进行遍历。

- 1) size 返回Map的元素个数
- 2) set 增加一个新元素，返回当前Map
- 3) get 返回键名对象的键值
- 4) has 检测Map中是否包含某个元素，返回 boolean值
- 5) clear 清空集合，返回undefined

```
<div class="mydiv"></div>
    <script>
        let map = new Map();

        function clickFn() {}
        map.set([1, 2, 3], 100);

        map.set(document.querySelector('.mydiv'), {
            innerHTML: '要显示的内容',
            display: 'none',
            click: clickFn,
            order: 2
        });

        map.set(null, 500);
        map.set(undefined, 800);
        /**
        console.log(map);
        console.log(map.get(null));
        console.log(map.has(undefined));
        console.log(map.has([1,2,3]));
        map.delete([1,2,3]);
        console.log(map);
        */
        map.forEach(m => console.log(m));
    </script>
```

Map 的属性和方法：

（1）size 属性

size属性返回 Map 结构的成员总数。

```
const map = new Map();
map.set('foo', true);
map.set('bar', false);

map.size // 2
```

（2）Map.prototype.set(key, value)

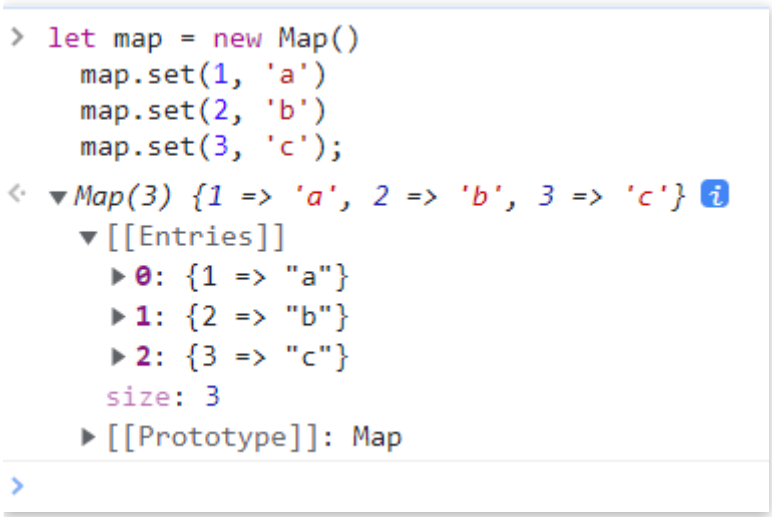
set方法设置键名key对应的键值为value，然后返回整个 Map 结构。如果key已经有值，则键值会被更新，否则就新生成该键。

```
const m = new Map();

m.set('edition', 6)        // 键是字符串
m.set(262, 'standard')     // 键是数值
m.set(undefined, 'nah')    // 键是undefined
```

set方法返回的是当前的Map对象，因此可以采用链式写法。

```
let map = new Map()
  map.set(1, 'a')
  map.set(2, 'b')
  map.set(3, 'c');
  //Map(3) {1 => 'a', 2 => 'b', 3 => 'c'}
```



（3）Map.prototype.get(key)

get方法读取key对应的键值，如果找不到key，返回undefined。

```
const m = new Map();

const hello = function() {console.log('hello');};
m.set(hello, 'Hello ES6!') // 键是函数

m.get(hello)  // Hello ES6!
```

（4）Map.prototype.has(key)

has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中。

```
const m = new Map();

  m.set('edition', 6);
  m.set(262, 'standard');
  m.set(undefined, 'nah');

```

```
m.has('edition')     // true
m.has('years')       // false
m.has(262)           // true
m.has(undefined)     // true
```

（5）Map.prototype.delete(key)

delete方法删除某个键，返回true。如果删除失败，返回false。

```
const m = new Map();
m.set(undefined, 'nah');
m.has(undefined)     // true

m.delete(undefined)
m.has(undefined)       // false
```

（6）Map.prototype.clear()

clear方法清除所有成员，没有返回值。

```
let map = new Map();
map.set('foo', true);
map.set('bar', false);

map.size // 2
map.clear()
map.size // 0
```

Map 遍历方法：

Map 结构原生提供三个遍历器生成函数和一个遍历方法。

- Map.prototype.keys()：返回键名的遍历器。
- Map.prototype.values()：返回键值的遍历器。
- Map.prototype.entries()：返回所有成员的遍历器。
- Map.prototype.forEach()：遍历 Map 的所有成员。

需要特别注意的是，Map 的遍历顺序就是插入顺序。

```
const map = new Map([
```

```
  ['F', 'no'],
  ['T',  'yes'],
]);

for (let key of map.keys()) {
  console.log(key);
}
// "F"
// "T"

for (let value of map.values()) {
  console.log(value);
}
// "no"
// "yes"

for (let item of map.entries()) {
  console.log(item[0], item[1]);
}
// "F" "no"
// "T" "yes"

// 或者
for (let [key, value] of map.entries()) {
  console.log(key, value);
}
// "F" "no"
// "T" "yes"

// 等同于使用map.entries()
for (let [key, value] of map) {
  console.log(key, value);
}
// "F" "no"
// "T" "yes"
```

2、ES6：严格模式

严格模式主要有以下限制：

- 变量必须声明后再使用

- 函数的参数不能有同名属性，否则报错
- 不能使用with语句
- 不能对只读属性赋值，否则报错
- 不能使用前缀 0 表示八进制数，否则报错
- 不能删除不可删除的属性，否则报错
- 不能删除变量delete prop，会报错，只能删除属性delete global[prop]
- eval不会在它的外层作用域引入变量
- eval和arguments不能被重新赋值
- arguments不会自动反映函数参数的变化
- 不能使用arguments.callee
- 不能使用arguments.caller
- 禁止this指向全局对象
- 不能使用fn.caller和fn.arguments获取函数调用的堆栈
- 增加了保留字（比如protected、static和interface）

```
 //启用严格模式
'use strict';

function fn() {
    //1，严格模式下不能给一个没有声明的变量赋值
    var b = 200;
    console.log(b);
}
fn();
//2，严格模式下不能 delete 一个变量
var c = 20;
c = null;
//3，严格模式下去调用一个全局函数，那么函数里面的this指向undefined
function f() {
    console.log(this);
}
f();
//4，构造函数不能当成普通函数来直接调用，必须new
function Person(name) {
     console.log(300);
     this.name = name;
}
new Person('ccc');
```

```
24
25 //5，函数的形参不能重名
26 function f1(a, a) {
27
28 }
```

3、ES6：Promise

Promise 是异步编程的一种解决方案

（1）对象的状态不受外界影响，有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）

（2）一旦状态改变，就不会再变，任何时候都可以得到这个结果，只有两种可能：从pending变为fulfilled和从pending变为rejected

回调地狱：

```
<script>
        //专业-->年级-->班级-->人数   模拟：ajax   到 服务器拿数据
        setTimeout(function () {
            console.log('获取专业数据');
            //获取到专业数据之后，去获取年级数据
            setTimeout(function(){
                console.log('获取年级数据');
                //获取到年级数据之后，去获取班级数据
                setTimeout(function(){
                    console.log('获取班级数据');
                    //获取到班级数据之后，去获取班级人数
                    setTimeout(function(){
                        console.log('获取到班级人数了');
                    }, 300);
                }, 600);
            }, 800);
        }, 1000);
    </script>
```

Promise认识：

ES6 规定，Promise对象是一个构造函数，用来生成Promise实例，Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject

- resolve：将Promise对象的状态从“未完成”变为“成功”（即从 pending 变为 resolved），在

异步操作成功时调用，并将异步操作的结果，作为参数传递出去

- reject：将Promise对象的状态从“未完成”变为“失败”（即从 pending 变为 rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去

```
//实例化 Promise 对象
        const p = new Promise(function (resolve, reject) {
            setTimeout(function () {

                // 成功调用resolve()处理
                let data = "数据读取成功";
                resolve(data);

                // 失败调用reject()处理
                let err = "数据读取失败";
                reject(err);

            }, 3000);
        });

        //调用 promise 对象的 then 方法
        p.then(function (value) {
            console.log(value);
        }, function (reason) {
            console.error(reason);
        });
```

Promise-then方法：

Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数。

```
<script>
        /**
            1，new本身是创建实例化对象，是同步的；
            2，Promise构造函数本身是同步的：承诺已经许下了；
            3，then是异步的：promise状态的改变才会触发then的回调函数；
        */
        let p1 = new Promise(function (resolve, reject) {
            console.log(0);
```

```
            //resolve：把状态从pending 变成  fulfilled
            //reject：把状态从pending 变成 reject
            setTimeout(function () {
                if (Math.random() > 0.5) {
                    resolve()
                } else {
                    reject();
                }
            }, 2000);
        });
        console.log(p1);
        //接收promise状态的改变
        console.log(2);
        for (let i = 0; i < 1000000000; i++);

        p1.then(function () {
            //pending---->fulfilled  成功  异步
            console.log('状态从pending变为fulfilled');
        }).catch(function () {
            //pending---->rejected  失败  异步
            console.log('状态从pending变为rejected');
        });
        console.log(1);
    </script>
```

Promise：状态改变，传参

```
<script src="./js/items.js"></script>
    <script>
        /**
            1,resolve是js解释器内置的方法，用于改变promise的状态为fulfilled；
            2，当状态改变为fulfilled之后，会触发then这个方法；
            3，resolve这个方法和then的回调函数不是一个东西；
        */
        let p1 = new Promise(function (resolve, reject) {
            console.log(1);
            setTimeout(function () { //模拟我花了2s的时间去服务器拿数据：ajax
                if (Math.random() > 0.4) {
```

```
                    //改变promise为fulfilled
                    resolve(data);
                } else {
                    //改变promise为rejected
                    reject({
                        code: -1,
                        msg: '服务器挂了'
                    });
                }
            }, 2000);
        });

        p1.then(function (data) {
            //业务处理-->把数据渲染到页面之上
            console.log(data);
        }).catch(function (err) {
        //catch 用于指定发生错误时的回调函数
        //then()方法指定的回调函数，如果运行中抛出错误，也会被catch()方法捕获。
            //容错处理--》
            console.log(err);
            alert(err.msg);
        }).finally(function () {
        //finally()方法用于指定不管 Promise 对象最后状态如何，都会执行的操作
            console.log('日志记录');
        });
    </script>
```

Promise.all（）方法：Promise.all()方法用于将多个 Promise 实例，包装成一个新的 Promise 实例

```
<script src="./js/items.js"></script>
    <script>
        let p1 = new Promise((res, rej) => {
            //res(1);
            setTimeout(function(){
                rej({code:-1, msg:'个人信息获取失败'});
            }, 5000);
        });

```

```
        let p2 = new Promise(function (resolve, reject) {
            resolve([100, 200, 300]);
        });

        let p3 = new Promise(res => {
            res({ a: 1, b: 2 });
        });

        let p = Promise.all([p1, p2, p3]);
        p.then(function (data) {
            console.log(data);
        }).catch(function(err){
            console.log(err);
        });
    </script>

//声明两个promise对象
        const p1 = new Promise((resolve, reject)=>{
            setTimeout(()=>{
                resolve('商品数据 - 1');
            },1000)
        });

        const p2 = new Promise((resolve, reject)=>{
            setTimeout(()=>{
                resolve('商品数据 - 2');
                // reject('出错啦!');
            },1000)
        });

        //调用 allsettled 方法
        const result = Promise.allSettled([p1, p2]);始终有返回结果
        // const res = Promise.all([p1, p2]);

        console.log(res);
```

Promise.race（）方法：Promise.race()方法同样是将多个 Promise 实例，包装成一个新的 Promise 实例；只要race()之中有一个实例率先改变状态，新Promise 实例的状态就跟着改变。

```
<script src="./js/items.js"></script>
    <script>
        let p1 = new Promise(function (resolve, reject) {
            setTimeout(function () {
                resolve(data);//data来自items.js文件
            }, 3000);
        });
        let p2 = new Promise(function (resolve, reject) {
            setTimeout(function () {
                resolve({
                    code:-1001,
                    msg:'加载超时，请点击空白处刷新...'
                });
            }, 8000);
        });
        Promise.race([p1, p2]).then(function(data){
  //只要p1、p2之中有一个实例率先改变状态，新的 Promise 实例的状态就跟着改变
            if(data.code === 200){
                console.log('把数据渲染到页面之上', data.itemsArr);
            }else{
                document.write(data.msg);
            }
        });
    </script>
```

4、ES6：遍历器

for...of循环：

```
        //循环字符串
        let str = 'abc';
        for (let s of str) {
            console.log(s);
        } //a  b   c

        //循环数组
        let arr = [1, 2, 3];
```

```
        for (let item of arr) {
            console.log(item);
        }//1  2  3

        //对象
        let obj = {
            a: 1,
            b: 2,
            c: 3
        };
        for (let [k, v] of Object.entries(obj)) {
            console.log(k, v);
        }// a 1
         // b 2
         // c 3

        //set
        let set = new Set([1, 2, 3]);
        console.log(set); //Set(3) {1, 2, 3}
        for (let s of set) {
            console.log(s);// 1  2   3
        }
        for (let s of set.entries()) {
            console.log(s);//(2) [1, 1]   //(2) [2, 2]  //(2) [3, 3]
        }

        //map
        let map = new Map();
        map.set(null, 'value1');
        map.set(undefined, 'value2');
        map.set(true, 'value3');
        map.set(false, 'value4');
        map.set({
            a: 1,
            b: 2
        }, 'value5');

        for (let m of map) {
            console.log(m);
        }
```

5、ES6：Generator生成器函数

Generator 函数是 ES6 提供的一种异步编程解决方案，Generator 函数是一个状态机，封装了多个内部状态，执行 Generator 函数会返回一个遍历器对象，也就是说，Generator 函数除了状态机，还是一个遍历器对象生成函数。，Generator 函数是一个普通函数，但是有两个特征。

- function关键字与函数名之间有一个星号；
- 函数体内部使用yield表达式，定义不同的内部状态

说明：

- * 的位置没有限制
- 生成器函数返回的结果是迭代器对象，调用迭代器对象的 next 方法可以得到 yield 语句后的值
- yield 相当于函数的暂停标记，也可以认为是函数的分隔符，每调用一次 next 方法，执行一段代码
- next 方法可以传递实参，作为 yield 语句的返回值

```
 //在function和函数名之间加上 *,并没有明确的指定位置
function* generatorFn(){
    console.log(100);
    yield 600; //产出
    for(let i = 0; i<3; i++) console.log(i);
    yield {username:'人才', age:19};
    yield [1,2,3];
    //有没有返回值需要注意一下
}
/*
let r = generatorFn();
   console.log(r);
   //让指针指向下一个yield
   console.log(r.next());
   console.log(r.next());
   console.log(r.next());
*/
//generator函数返回的是一个遍历器对象
let r1 = generatorFn();
for ( let g of r1){
    console.log(g);
    }
```

Generator.next：传参

yield表达式本身没有返回值，或者说总是返回undefined。next方法可以带一个参数，该参数就会被当作上一个yield表达式的返回值。

```
function* gFn(x){
    let y = yield x + 1;
    y = y / 4;
    console.log(y);
    let z = yield  y;
    return x + y + z;
}
   let g = gFn(10);
   console.log(g.next()); //{value: 11, done: false}
   console.log(g.next('')); // {value: 0, done: false}
   console.log(g.next(10)); // {value: 20, done: true}
```

Generator：异步处理

```
//解决回调地狱问题
        //请求数据的函数
        function getJSON() {
            const data = [{
                    name: '计算机学院',
                    id: 1001
                },
                {
                    name: '财经学院',
                    id: 1007
                }, {
                    name: '外贸学院',
                    id: 1009
                }
            ];
            return new Promise(function (resolve, reject) {
                setTimeout(function () { //模拟一个异步处理
                    resolve({
                        code: 1,
                        msg: '成功',
                        data
```

```
                    });
                }, 3000);
            });
        }
        //数据渲染的函数
        function renderData(data) {
            console.log('对数据进行渲染', data);
            return [1, 2, 3];
        }
        //其他的一些相关操作
        function otherOper(show) {
            console.log(show);
            return true;
        }

        function showBox(ifshow) {
            console.log(ifshow);
        }
        //generator 是异步解决方案
        function* json() {
            let data123 = yield getJSON();
            let data1 = yield renderData(data123);
            let data2 = yield otherOper(data1);
            yield showBox(data2);
        }

        let d = json(); //返回一个容器
        //console.log(d.next());//yield getJSON(); 需要注释掉
        d.next().value.then(function (data) {
            let r = d.next(data);
            let r1 = d.next(r.value);
            d.next(r1.value);
        });
```

6、ES6：async函数

async函数就是将 Generator 函数的星号（*）替换成async，将yield替换成await

async 函数的语法：

```
async function fn(){

}
```

async 函数的返回值：

- 返回的结果是一个 Promise 类型的对象，返回的结果就是成功 Promise 对象
- 返回的结果如果是一个 Promise 对象，具体需要看执行resolve方法还是reject方法
- 抛出错误，返回的结果是一个失败的 Promise

async 函数的案例：

```
//async 函数
async function fn() {
    return new Promise((resolve, reject) => {
        resolve('成功的数据');
// reject("失败的错误");
  });
}
const result = fn();
//调用 then 方法
result.then(value => {
    console.log(value);
}, reason => {
    console.warn(reason);
});
```

await 表达式

- async 和 await 两种语法结合可以让异步代码像同步代码一样
- await 表达式的注意事项：
- await 必须写在 async 函数中
- await 右侧的表达式一般为 promise 对象
- await 返回的是 promise 成功的值
- await 的 promise 失败了, 就会抛出异常, 需要通过 try...catch 捕获处理

await 表达式的语法案例：

```
//创建 promise 对象
const p = new Promise((resolve, reject) => {
    resolve("用户数据");
```

```
//reject("失败啦!");
})
//await 要放在 async 函数中.
async function fun() {
   try {
       let result = await p;
        console.log(result);
   } catch (e) {
        console.log(e);
   }
}
//调用函数
fun();
```

await 表达式的案例：async与await封装AJAX请求

```
// 发送 AJAX 请求, 返回的结果是 Promise 对象
function sendAJAX(url) {
    return new Promise((resolve, reject) => {
        //1. 创建对象
        const x = new XMLHttpRequest();
        //2. 初始化
        x.open('GET', url);
        //3. 发送
        x.send();
        //4. 事件绑定
        x.onreadystatechange = function () {
            if (x.readyState === 4) {
                if (x.status >= 200 && x.status < 300) {
                    resolve(x.response);//成功
                } else {
                    reject(x.status);//失败
                }
            }
        }
    })
}
// async 与 await 测试
async function fun() {
```

```
    //发送 AJAX 请求 1
    let joke = await sendAJAX("https://api.apiopen.top/getJoke");
    //发送 AJAX 请求 2
    let tianqi = await sendAJAX('https://www.tianqiapi.com/api/?version=v1&city=%E5%8C%97%E4%BA%AC&appid=23941491&appsecret=TXoD5e8P')
    console.log(joke);
    console.error(tianqi);//为了区别数据，我这里用红色的error输出
}
// 调用函数
fun();
```

async函数是Promise的语法糖：

```
//解决回调地狱问题
        //请求数据的函数
        function getJSON() {
            const data = [{
                    name: '计算机学院',
                    id: 1001
                },
                {
                    name: '财经学院',
                    id: 1007
                }, {
                    name: '外贸学院',
                    id: 1009
                }
            ];
            return new Promise(function (resolve, reject) {
                setTimeout(function () { //模拟一个异步处理
                    resolve({
                        code: 1,
                        msg: '成功',
                        data
                    });
                }, 5000);
            });
        }

        function getJSON1() {
```

```
        const data = [{
                name: '计算机学院',
                id: 1001
            },
            {
                name: '财经学院',
                id: 1007
            }, {
                name: '外贸学院',
                id: 1009
            }
        ];
        return new Promise(function (resolve, reject) {
            setTimeout(function () { //模拟一个异步处理
                resolve({
                    code: 1,
                    msg: '成功',
                    data
                });
            }, 3000);
        });
    }

    //async语法糖
    async function json() {
        console.log(100);
        let data123 = await getJSON();
        console.log(data123);
        let data1 = await getJSON1();
        console.log(data1);
        console.log(200);
    }
    json();

    async function reg() {
        console.log(100);
    }
    reg();

    function fn() {
```

```
        await getJSON();
    }
```

7、Map购物车案例

html代码

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ES6</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        body {
            font-family: '微软雅黑', Arial, sans-serif;
        }

        li {
            list-style: none;
        }

        .items {
            width: 800px;
            margin: 30px auto;
        }

        .items>li,
        .cars>ul>li {
            display: flex;
            justify-content: space-between;
        }
```

```
33
34          .items>li>.l>img {
35              width: 300px;
36              height: 300px;
37          }
38
39          .items>li>.r {
40              margin-left: 20px;
41          }
42
43          .items>li>.r>div:first-child {
44              font-weight: bold;
45              font-size: 18px;
46          }
47
48          .items>li>.r>div:nth-child(3),
49          .cars>ul>li>.r>div:nth-child(4) {
50              border-radius: 5px;
51              width: 120px;
52              height: 42px;
53              background-color: #000;
54              color: #fff;
55              text-align: center;
56              line-height: 42px;
57              font-size: 16px;
58              cursor: pointer;
59              margin-top: 50px;
60          }
61
62          .cars {
63              width: 400px;
64              background-color: rgb(225, 243, 63);
65              position: fixed;
66              bottom: 0;
67              right: 0;
68              max-height: 100vh;
69              overflow-y: auto;
70          }
71
72          .cars>ul>li {
```

```
            margin-bottom: 15px;
            padding: 20px;
            border-bottom: 1px solid rgb(180, 23, 23);
        }

        .cars>ul>li>.l>img {
            width: 120px;
            height: 120px;
        }

        .cars>ul>li>.r {
            margin-left: 20px;
        }

        .cars>ul>li.total {
            height: 50px;
            line-height: 50px;
            background-color: #000;
            color: #fff;
            padding: 0 50px;
        }

        .cars>ul>li>.r>div:nth-child(4) {
            margin-top: 10px;
            height: 32px;
            line-height: 32px;

        }
    </style>
</head>

<body>
    <ul class="items"></ul>

    <div class="cars">
        <ul class="carsul">
            <li>购物车为空</li>
        </ul>
        <ul class="totl">
```

```
            <li class="total">
                <div class="l">购买数量：<span class="tnum">0</span></div>
                <div class="r">总价：<span class="tmoney">0</span></div>
            </li>
        </ul>
    </div>

    <script src="./js/items.js"></script>
    <script src="./js/car.js"></script>
</body>

</html>
```

item商品

```
let itemsArr = [
    {
        id: 10032430666486,
        title: 'LEDE雪纺连衣裙女装2022夏季新款修身显瘦超仙气质中长款小个子碎花裙子夏天',
        preview: 'https://img11.360buyimg.com/n1/jfs/t1/190071/16/9509/88150/60cf0957E11d2d2c5/d8d321ebc01e736e.jpg',
        price: 168.00,
        sku: 60
    },
    {
        id: 10031962536266,
        title: '短袖t恤男生夏季纯棉T恤上衣服饰圆领男士半袖上衣服打底小汗衫半袖男装白色小t',
        preview: 'https://img11.360buyimg.com/n1/jfs/t1/96529/29/21044/199076/6204bbcaEe03d4d0d/43503c8487dd3ed2.jpg.avif',
        price: 59.00,
        sku: 89
    },

    {
        id: 10032595113039,
        title: '纯信纯棉t恤女短袖2021新款短袖女夏季韩版宽松显瘦胖mm百搭大码休闲女装上衣',
        preview: 'https://img14.360buyimg.com/n1/jfs/t1/196500/26/8481/81820/60cac69cEb1f91b2d/44cec7b01593a273.jpg',
```

```
        price: 49.00,
        sku: 93
    },
    {
        id: 70733221480,
        title: '【新款上市 】男士短袖T恤年轻小伙大学生男装衣服20-30岁25成年人纯棉夏装少男 灰色715 XL',
        preview: 'https://img10.360buyimg.com/n1/jfs/t1/126699/37/5189/109229/5eeb3dbeEf8c25fb9/1a69b8f56b7c8af0.jpg.avif',
        price: 118.00,
        sku: 516
    },
    {
        id: 10046972055053,
        title: '领季连衣裙女装2022年新款连衣裙夏时尚百搭韩版运动套装女修身显瘦洋气减龄',
        preview: 'https://img13.360buyimg.com/n1/jfs/t1/70692/12/17609/68583/627005faE1c62ffde/ea4b7cea6a2ade57.jpg.avif',
        price: 138.00,
        sku: 1056
    },
    {
        id: 10051721974439,
        title: '西域 骆驼男装【上市 】男士短袖T恤年轻小伙大学生男装衣服-30岁25成年人夏装少',
        preview: 'https://img14.360buyimg.com/n1/jfs/t1/75324/21/18338/152078/6278c09cE12ea7bc4/62efbadded7b814c.jpg.avif',
        price: 116.00,
        sku: 56
    },
    {
        id: 10046421982031,
        title: '妍莉芬 连衣裙女装2022年新款夏季时尚女装职业气质通勤收腰显瘦假两件装衬衫小',
        preview: 'https://img11.360buyimg.com/n1/jfs/t1/120252/15/22179/162268/622f5006E2a5acc67/2f58742ecbfe11d8.jpg.avif',
        price: 129.00,
        sku: 106
    }
];
```

car购物车

```
(function () {
    //1 把商品信息显示在页面上
    let liArr = itemsArr.map(item => `<li>
                    <div class="l"><img src="${item.preview}" alt=""></div>
                    <div class="r">
                        <div>${item.title}</div>
                        <div>价格：${item.price}</div>
                        <div data-id="${item.id}" class="add">加入购物车</div>
                    </div>
                </li>`);
    //liArr经过map处理后是数组，使用join方法转为字符串
    document.querySelector('.items').innerHTML = liArr.join('');

    //2 给“加入购物车”按钮绑定点击事件
    let addBtn = document.querySelectorAll('.add');
    console.log(addBtn);
    //循环的方式给每个按钮绑定点击事件
    for (let i = 0; i < addBtn.length; i++) {
        addBtn[i].addEventListener('click', function () {
            cars.add(this.dataset.id);
        })
    }

    //3 购物车对象--->用cars来描述
    let cars = {
        //使用map处理商品信息
        items: new Map(),

        //4 点击动作(函数)
        add: function (id) {
            //该商品已经加入购物车,数量自加1
            if (this.items.has(id)) {
                this.items.get(id).count++;
                //否则获取商品
            } else {
                let itemId = this.getItem(id);
                itemId.count = 1;
                this.items.set(id, itemId);
```

```
            }
            //显示商品添加后的信息
            this.show();
        },

        //5 根据id获取商品信息
        getItem: function (id) {
            return itemsArr.find(p => id / 1 === p.id);
        },

        //6 根据id移出商品
        delete: function (id) {
            this.items.delete(id);
            this.show();
        },

        //7 点击后商品信息展示
        show: function () {
            let html = '';
            this.items.forEach(function (item, ind) {
                //console.log(item, ind);
                html += `<li>
                            <div class="l">
                                <img src="${item.preview}" alt="">
                            </div>
                            <div class="r">
                                <div>${item.title}</div>
                                <div>${item.price}</div>
                                <div>购买数量：${item.count}</div>
                                <div data-id="${item.id}" class="remove">移出购物车</div>
                            </div>
                        </li>`;
            });
            document.querySelector('.carsul').innerHTML = html;
            //计算商品数量和价格的总和
            this.total();
            //给移出按钮绑定点击事件
            this.yiChu();
        },
```

```
78
79          //8 商品数量和价格总和 方法
80          total: function () {
81              let num = 0, money = 0;
82              this.items.forEach(function (item, ind) {
83                  num += item.count;
84                  money += item.count * item.price;
85              });
86              document.querySelector('.tnum').innerHTML = num;
87              document.querySelector('.tmoney').innerHTML = money.toFixed(2);
88          },
89
90          //9 商品移出
91          yiChu: function () {
92              let _this = this;
93              let removeBtn = [...document.querySelectorAll('.remove')];
94              removeBtn.forEach(function (btn) {
95                  btn.onclick = function () {
96                      _this.delete(this.dataset.id);
97                  }
98              });
99          }
100
101     };
102 })();
```